

Performance and security exploration of aggressor-focussed row-migration using row swap approach and it's enhancements

Hena Naaz

Abstract—With increasing DRAM scaling, the threat of Row Hammer Attacks are increasing rapidly. In this project, we span the implementation of RRS/SSRS in a more performance efficient model. This row-migration technique has been proposed as a means of mitigating Row Hammer Attacks, but its security, performance overhead, and complexity in design remains a concern. In this paper, we present an optimized implementation of RRS/SSRS that achieves higher performance efficiency compared to previous work and presents a secure system. The goal of the project is to optimize the row-migration implementation to maintain low overhead. This work contributes to the ongoing effort to develop scalable and secure techniques for mitigating Row Hammer Attacks in DRAM systems.



1 INTRODUCTION

Row Hammer attacks pose a significant threat to the security and reliability of DRAM systems, and as DRAM scaling continues, the risk of these attacks is only increasing. Several mitigation techniques have been proposed to address this issue, but these techniques often come with performance overhead, which can limit their practicality.

Due to the nature of attacks on victim-focused mitigation, the focus has been shifted on aggressor-focused mitigation. One promising technique for mitigating Row Hammer attacks is the extension of Randomized Row Swap (RRS) / Scalable and Secure Row Swap (SSRS) approach with a more effective tracker, which has been shown to achieve a more efficient design while also offering promise for future scalability. However, the performance implications of this technique have yet to be fully addressed.

In this paper, we aim to address these issues by investigating the RRS technique and exploring ways to optimize its performance. Specifically, we have implemented RRS with the baseline misra-gries tracker to compare with baseline (victim refresh method) and evaluated its performance using a range of benchmarks and workloads. We have also explored different approaches for improving the performance of RRS, including extending the design with SRAM based trackers and investigating design enhancements for reducing memory overheads across swap handling. [2] [3]

The project aims to contribute to the ongoing effort to develop more secure and efficient aggressor-focused techniques for mitigating Row Hammer attacks in DRAM systems.

2 BACKGROUND

Row Hammer attacks are a well-known and growing threat to the security and reliability of DRAM systems. Various mitigation techniques have been proposed to address this issue, including ECC memory, Target Row Refresh (TRR), Row Clone, and Randomized Row Swap. While these techniques have shown promise, they also have limitations such as high cost, limited scalability, and limited effectiveness against sophisticated Row Hammer attacks. [3]

The Scalable and Secure Row-Swap (SSRS) technique was proposed as a means of addressing these limitations. This technique utilizes a hybrid row swap algorithm to

randomly swap rows in DRAM, making it difficult for attackers to predict which rows will be affected by a Row Hammer attack. [2] Despite these benefits, however, SSRS also has limitations in the reliance on probabilistic nature of mitigation and a software-based security algorithm, which may not be accurate or effective in all scenarios.

Another limitation of RRS is the potential impact on memory access latency and bandwidth due to the additional overheads introduced by the hybrid row swap algorithm that needs to maintain history of swap for un-swaps. This overhead can reduce the overall performance of the system and limit its practicality in certain scenarios. Therefore, optimizing the algorithm and its implementation is crucial to minimize these overheads and improve the overall performance of the SSRS technique.

In this paper, we aim to build upon the existing research on RRS/SSRS by exploring ways to improve its performance and security. There are 3 approaches that have been practiced for this, targeting the detection, mitigation, and performance of the design. These are described in the design section of the paper.

3 METHODOLOGY

There are also limitations of the discussed Row-Swap techniques, that includes the potential impact on memory access latency and bandwidth due to the additional overheads introduced by the hybrid row swap and un-swap operations. To address these the project could explore the optimization of the algorithm and its implementation.

This involved techniques such as tracker modification, tracker and table data structure optimizations, and smarter scheduling of memory operations to minimize the overheads, thereby, improving the performance of the technique. Furthermore, the safety aspect of the design can be discussed for better row-migration schemes.

For these experiments, there are following three hypothesis taken into picture, that shall be explained in detail in the proposed design section:

- 1) RRS/SSRS design will improve in performance with an advanced tracker like Hydra [1] where SRAM based storage as hybrid tracker is also practiced.
- 2) RRS/SSRS design will improve by replacing the row indirection table that gets exploded and cause

memory overhead with a tracker map that keep pointer to the swap for un-swaps..

- 3) The design can be made more secure by replacing the randomization approach with a isolation based approach with a tracker map that keep pointer to the migrated row. [4]

The scope of this project has tested these hypothesis in a simulation environment to study the memory overheads and performance implications of the proposed methodology.

In order to perform these experiments, the following configuration of Simplesim simulator was used:

Criterion	Used methodology
Simulator	Simple Sim (CS7292 Lab1 simulator)
Benchmarks	(SPEC) bzip, bwaves, Gems, cactus, mcf, zeusmp
Configuration	4 CPU OoO cores and multi-level set-associative caches
Memory config	1 Channel * 1 rank * 16 banks Memory Size: 16 GB, Row Size: 8 KB
Metrics	Number of accesses, number of migrations, total sys cycles, total mitigations.

Fig. 1. The methodology used to perform the stated experiments.

4 DESIGN EXPLORATION

4.1 Overview

The proposed design extends the Randomized Row-Swap (RRS) technique by replacing the Misra-Gries tracker with modifications based on the ongoing and proven research in the field of Row Hammer Mitigation. It also tries to reduce the memory overhead of storing the tracker and indirection table by exploring integrated structures.

There are certain aspects of the RRS design that are of particular interest to the project. The row indirection table is used to map the physical addresses of DRAM rows to virtual addresses, which are then used to perform row swaps. The optimization and better management of this table to reduce the overhead and improve the overall performance of the technique is another track explored. [2]

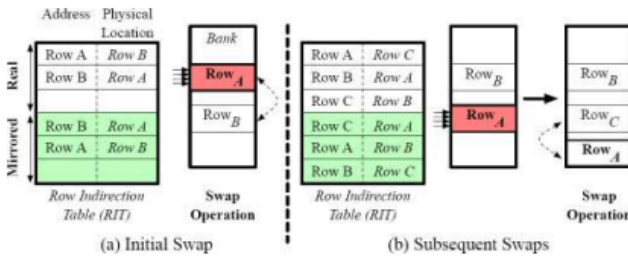


Fig. 2. Row Indirection Table (real and mirrored) provides indirection to the rows involved in the swap operations in SSRS.

The swap counter is used to track the number of row swaps that have been performed in each bank. This is to keep an eye on the total number of migrations that have occurred to iterate the threshold changes and data movement relation. When a row swap is performed, the swap counter is also incremented for the corresponding bank. This information is used to determine when it is necessary

to perform a full refresh of the DRAM bank, which is a key step in mitigating Row Hammer attacks. The project plans to optimize the management of this counter to reduce the overhead and improve the scalability of the technique. Further, the goal is to explore row-migration reduction and security improvement leveraging other aggressor-focused mitigation schemes.

4.2 SRAM based Row Hammer tracker

It is known that the Hydra tracker, which is a hybrid tracker between misra-gries (used in RRS/SSRS) and CRA tracker, is a more efficient and accurate tracker that uses a set of counters to keep track of the number of times each row has been accessed. This allows the system to detect and mitigate Row Hammer attacks more effectively, without relying on a software-based security algorithm.

Graphene Tracker using Misra Gries might have slightly lower accuracy metrics due to its simpler tracking mechanism, leading to potential under-detection or over-detection of Row Hammer vulnerable rows. Hydra achieves higher detection accuracy compared to Graphene due to its hybrid approach, combining algorithmic tracking for initial identification of aggressor rows and exact tracking for confirmation and precise counting. [1]

Hydra leverages SRAM-based structure that tracks aggregated counts at the granularity of a group of rows and a per row tracker stored in the DRAM-array which can track arbitrary number of rows. It was observed that with Hydra tracker, the main memory overhead for tracker structure has seen a reduction in magnitude. However, the overall memory overhead doesn't benefit from this change due to the larger overhead coming from the row-indirection table.

Although the perspective of SRAM based trackers is encouraging, there are still potential of attacks by leveraging cache side channel attacks and probabilistic nature of swaps. We therefore, investigated the design further and modified where needed. .

In the next exploration, the Hydra tracker in the RRS design was modified to extend the existing row indirection table and swap counters in the RRS design to improve the performance and scalability of the technique.

4.3 Tracker and Indirection Design Enhancement

In the previous section, it was observed that despite enhancing the design with a better tracker, the overall memory overhead didn't see much changes. The magnitude of the overall memory overhead is determined by the row-indirection table employed. Therefore, the project was modified to focus on reducing the data structure complication, in terms of maintaining independent structures for tracking the rows and then to keep a table for identifying swapped row pairs.

One way of achieving reduced storage space is to combine the row-indirection table with the counter and replacing the pair of swap operations with per row swapped address. This led to addition of extra entry equal to the size of one row address.

This led to increase in the overall lookup because now memory has to find the complimentary swap pair through constant look-up in the same table while tracking is in progress. It was also observed that one entry per bank, ended up with a similar memory overhead as before since the decrease in the size of one entry is still not enough to compensate for the fact that despite a decreased threshold of $Trh/6$, the total number of entries inside the tracker is much higher compared to total entries needed for just swapped pairs.

4.4 Accelerated Swap Operations

The design proposal for the accelerated swap operation involves dividing the memory system into multiple swap groups, each containing a set of DRAM banks that can be swapped simultaneously. To implement the accelerated swap operation, we propose a hybrid row swap algorithm that combines the use of the row indirection table and bank swap counters with the parallelism-based approach. The hybrid row swap algorithm works as follows: when a Row Hammer attack is detected, the Swap Group Manager selects two swap groups that contain the affected row and schedules the swap operations.

This design scheme was not effective due to swap group scope limiting the overall random address space, causing it further easy to probabilistic determination of the swapped pair. Due to above design explorations, it was observed that the major concerns lies in the randomized nature of mitigation. Therefore, the goal is to eliminate the randomized nature of mitigation. This exploration led to leveraging row quarantine region scheme proposed by AQUA. [4]

5 PROPOSED DESIGN

The RRS technique aims to mitigate Row Hammer attacks by using an aggressor-focused scheme by adding an indirection table. However, the technique still relies on randomization, which may not be effective against sophisticated attacks.

By evaluating the approaches mentioned in the proposed design section, it is concluded that randomization based approach is not as effective to be called a fool-proof design. Therefore the proposal involves reducing the dependencies on buffers to keep the mapping for Row indirection table, and add a quarantine region to create isolation for row-migration.

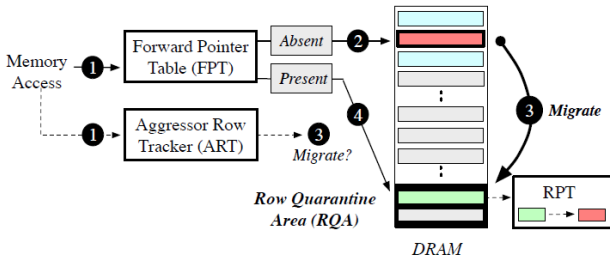


Fig. 3. AQUA design using FPT and RPT alongwith the row tracker.

The proposal is inspired by the AQUA design that presents a forward pointer and reverse pointer look-up tables. By observing the previous designs and experiments, it is observed that the tracker when combined with Hydra, which is a SRAM-based tracker, the DRAM memory overhead is improved and also the overall data movement. [4]

The indirection table has now been replaced by a pointer scheme where there is no need to maintain per bank structures. Now, with Hydra tracker, the total overhead of main memory operations has seen a significant drop. There is also a drop in the overall number of migrations across memory when it is compared with not just RRS but also RRS Hydra modifications.

6 RESULTS

Looking at the results from approach 1, there was no significant change in the performance of the design since even

with a hybrid tracker, the design was still using a global mapping table to store row indirection information leading to Fig. 4. A result of comparison between the designs with and without SRAM trackers shown in Fig. 5 highlights that as expected the hydra tracker due to accuracy will incur lesser row-migrations but it is not significant enough.

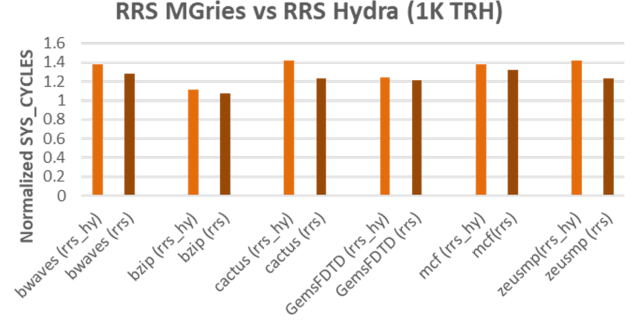


Fig. 4. Comparison between the normalized cycles RRS MGries vs RRS Hydra for benchmarks at 1024 Trh.

Scheme	Threshold	Migrations
rrs	1024	752665
rrs_hy	1024	26790
rrs	2048	427515
rrs_hy	2048	13312

Fig. 5. Comparison between the row-migration in RRS with Misra-gries tracker versus RRS with Hydra tracker.

Furthermore, the approach 2 showed a slight increase in the memory overhead by merging the row indirection table with the tracker per bank and adding pointers for isolation. Since the overall gain in one entry element doesn't recover since it is adding an extra row address size value for tracking, even if the threshold hasn't reached.

The qualitative and quantitative observations from these designs showed no significant performance improvement, around 1%. The experiment done on RRS design by reducing the data structure redundancy is that despite RIT integrated with Hydra tracker, the design is still probabilistic design. Therefore, it cannot be reliable for future use.

The quantitative analysis of the tracker and row-indirection table shows that indirection table is incurring order of magnitude greater overhead than tracker. When the tracker is changed to Hydra, the memory overhead caused by RIT is still present, and the number of entries increasing double folds as threshold decreases, Fig. 6.

The final design proposal presents AQUA with hydra tracker, where AQUA has shown better performance in terms of number of cycles as well as reduced number of row migrations, Fig 7.

The non-random row-migration for Row Hammer Attack mitigation has shown considerable improvement in the number of row-migrations, and that has improved further with Hydra tracking, Fig 8. This is also indicative of the fact that lesser number of row-migrations will incur less delay due to reduced memory access for the row-migrations. In this, it was observed that in order to conclude this, there was a need to find a benchmark using sufficient number of operations to exploit the RQA data structure with forward and reverse pointer tables, which was attained clearly in bzfp. When comparing traces in a 4 core simulation, on bzfp

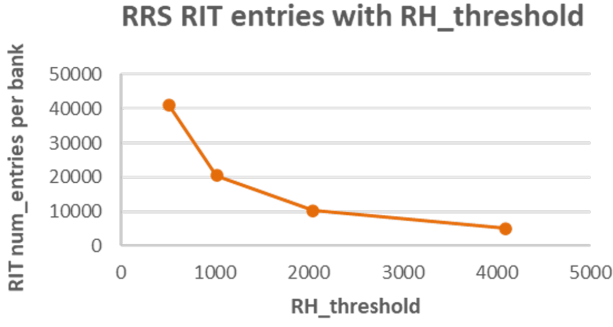


Fig. 6. The number of entries in RIT with threshold.

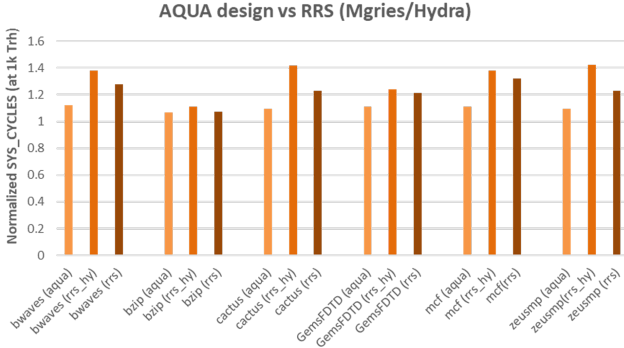


Fig. 7. Normalized cycles in AQUA implementation compared to RRS with MGries and Hydra tracker.

it gives a reduction on row-migrations by 7.5 times and on cactus traces by 3 times.

While we have discussed the gains, it is important to note that overall memory overhead has reduced since the per bank RIT has been replaced by a RQA table. Additionally, now because the order of magnitude in memory overhead between the tracker and RQA structure is comparable, the Hydra tracker has incurred 6 times memory overhead reduction.

Moreover, the significant gains with AQUA Hydra design has also shown some minor overheads. In terms of total memory activation, some benchmarks like bwaves and bzip have shown minor increase compared to victim-

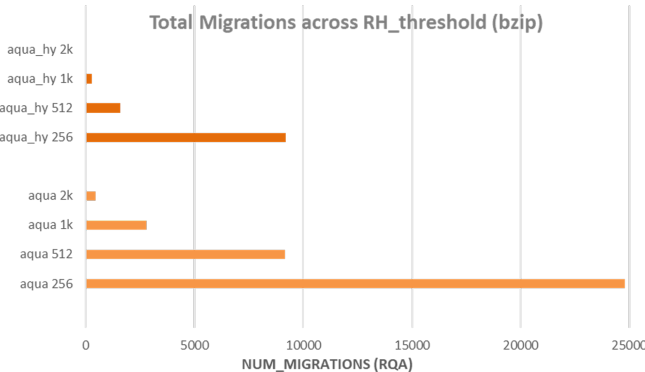


Fig. 8. Comparison between the row-migration in SRS versus AQUA with hydra.

focused mitigation. When it comes to total system cycles, benchmarks like cactus and mcf have shown significant increased in cycle for Hydra extension compared to AQUA design using MGries, Fig 9.

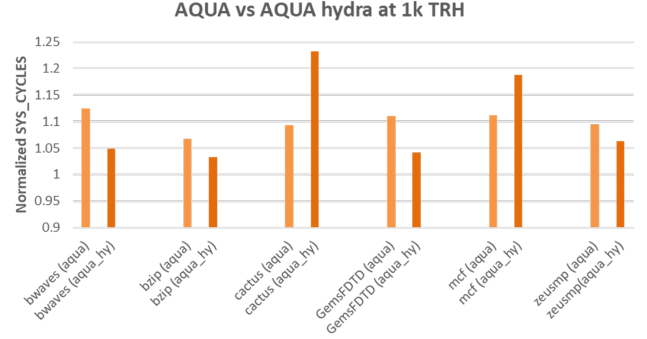


Fig. 9. Comparison between the normalized cycles at 1k threshold in AQUA MGries vs Hydra.

7 CONCLUSION

In this project, we proposed an extension to the row-swap technique for mitigating Row Hammer attacks in memory systems. Our proposed design aims to address the limitations of the RRS technique and improve its performance and scalability without requiring any additional memory overhead.

Unlike RRS and SRS, AQUA does not rely on randomization, which is a potential security threat. By integrating the Hydra tracker and modifying the design, we improved the efficiency of design. Our experiments showed that AQUA with non-random row-migration outperformed RRS and SRS in terms of the number of row migrations and security. Isolation-based techniques like AQUA have the potential to provide better security than randomization-based techniques for row hammer mitigation.

The future scope of the research indicated potential exploration on the redundancies in the FPT and RPT and the lazy evictions. The total number of memory access can be lowered down in the AQUA design by flagging off the quarantine addresses in the tracker to point to the actual row address. Furthermore, the challenge lies in evaluation where it would be a great idea to get control on benchmark to be able to exploit more tendencies on design by looking at the actual assembly code or instruction sequence in place.

REFERENCES

- [1] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: Enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 699–710. [Online]. Available: <https://doi.org/10.1145/3470496.3527421>
- [2] G. Saileshwar and P. J. Nair, "Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems," *arXiv:2212.12613*, 2023.
- [3] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized row-swap: Mitigating row hammer by breaking spatial correlation between aggressor and victim rows," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1056–1069. [Online]. Available: <https://doi.org/10.1145/3503222.3507716>
- [4] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 108–123.